Automatic vector generation guided by a functional metric

I. Ugarte, P. Sanchez

I. Ugarte, P. Sanchez, "Automatic vector generation guided by a functional metric," Proc. SPIE 8067, VLSI Circuits and Systems V, 80670U (3 May 2011); doi: 10.1117/12.887494

SPIE.

Event: SPIE Microtechnologies, 2011, Prague, Czech Republic

Automatic vector generation guided by a functional metric

I. Ugarte, P. Sanchez
University of Cantabria, Cantabria, Spain

ABSTRACT

Verification is still the bottleneck of the complex digital system design process. Formal techniques have advanced in their capacity to handle more complex descriptions, but they still suffer from problems of memory or time explosion. Simulation-based techniques handle descriptions of any size or complexity, but the efficiency of these techniques is reduced with the increase in the system complexity because of the exponential increase in the number of simulation tests necessary to maintain the coverage. Semi-formal techniques combine the advantages of simulation and formal techniques as they increase the efficiency of simulation-based verification. In this area, several research works have introduced techniques that automate the generation of vectors driven by traditional coverage metrics. However, these techniques do not ensure the detection of 100% of faults.

This paper presents a novel technique for the generation of vectors. A major benefit of the technique is the more efficient generation of test-benches than when using techniques based on structural metrics.

The technique introduced is more efficient since it relies on a novel coverage metric, which is more directly correlated to functional faults than structural coverage metrics (line, branch, etc.). The proposed coverage metric is based on an abstraction of the system as a set of polynomials where all system behaviours are described by a set of coefficients. By assuming a finite precision of coefficients and a maximum degree of polynomials, all the system behaviors, including both the correct and the incorrect ones, can be modeled. This technique applies mathematical theories (computer algebra and number theory) to calculate the coverage and to generate vectors which maximize coverage.

Moreover, in this work, a tool which implements the technique has been developed. This tool takes a C-based system description and provides the coverage and the generated vectors as output.

Keywords: Lagrange Interpolation, domain testing, coverage, mutation testing.

1. INTRODUCTION

Functional verification is the main task in the design process of complex system, which can account for 70% of the resources of the project. Several techniques exist to deal with the verification, from simulation techniques to formal techniques, passing through semiformal techniques. However, these techniques do not reduce the gap between the capacity of design and the capacity of verification.

The most common industrial technique is still simulation. However, its great inconveniences are in knowing when the system is sufficiently verified and in maximizing the probability of stimulating and detecting bugs at minimum cost in terms of time and computation resources. The main automatic technique to improve the verification capacity of the simulation is the coverage metric. For this reason, many research works have been focused on defining new coverage metrics, the ideal situation being when the metric has a direct correspondence to the design errors.

The automatic coverage metrics used in industrial tools are mainly the classical code-based coverage metrics derived from the software testing field [1]: line/block coverage, branch/conditional coverage, expression coverage and path coverage. However, these metrics only provide information about how the code has been stimulated. They are not enough to identify a stopping criterion.

Different research lines have attempted to improve the correlation between the coverage metrics and design errors. One approach is to evaluate the quality of the verification test benches with behavioral fault models based on the classic stuck-at-fault model [2, 3] or based on bit coverage of the variable, signal and port and the stuck-at of condition [4] or using the mutation fault coverage of software [5, 6] for hardware design verification [7].

VLSI Circuits and Systems V, edited by Teresa Riesgo, Eduardo de la Torre-Arnanz,
Proc. of SPIE Vol. 8067, 80670U · © 2011 SPIE · CCC code: 0277-786X/11/$18 · doi: 10.1117/12.887494

Another important approach is a tag (error) coverage method based on Hardware Description Language (HDL) coverage and observability information [8] and the control-oriented coverage metric that propagates error effects along all possible erroneous control-flow paths throughout processes [9].

Another strategy of software testing, domain analysis [10], is used to defined new coverage metrics [11, 12]. It is better than classical code-based coverage and it is focused on the detection of control flow errors of software. The work [11] evaluates the fault coverage in behavioral VHDL descriptions and [12] applies the new metric to measure the completeness and quality of the validation approach on several benchmark circuits and industrial circuits under design.

This work presents a new coverage that includes control and data flow. This coverage is based on mathematical theory and software testing which are described in section 3. As the system models are represented by polynomial functions (section 2), some mathematical theories can be applied to establish the number of vectors that completely define specific functionality. Moreover, it can be applied to improve some domain strategies in software testing. Section 4 presents the new metric and explains how the generation of test benches can be optimized. Finally, section 5 presents experimental results. An automatic program has been developed to obtain the test benches from C functions that model hardware and the final coverage. These results are compared with classical code-based coverage metrics and mutation testing. Finally, the conclusions are presented.

2. SYSTEM MODEL

In this paper, we assume that the hardware system is described as a set of statements that operate with integer data. These integer data are normally bounded, thus they can only take a finite set of values (typically 2^m two's complement values, where 'm' is the word bit number). Some basic operators are supported (addition, subtraction, multiplication, relational and logic operators) as well as 'if' control statements.

Concerning loop statements, a predefined number of iterations are unrolled in a similar way to the time frame expansion in ATPG and Bounded Model Checking [16]. The bit-level logic operations are converted into integer expressions.

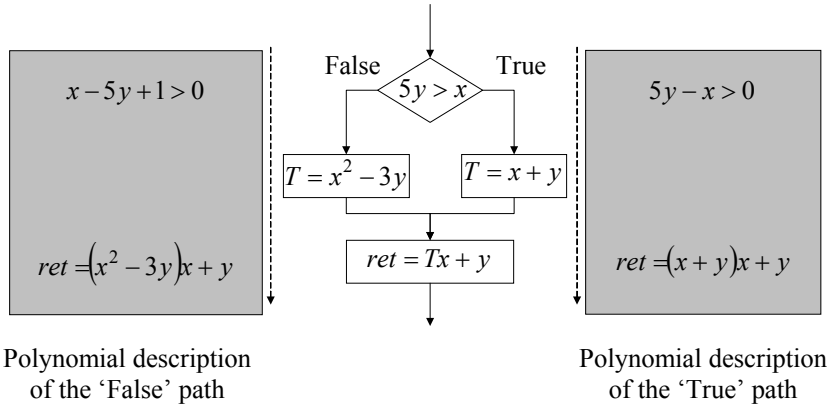


Fig. 1. Polynomial descriptions of a simple example.

With the previously commented consideration, every execution path in the system description can be modeled with polynomials (which model the system behavior) and a set of constraints, which model the if-statement conditions. A *if* statement is a constraint that is converted into a polynomial inequality or equality. A simple example is shown in Fig. 1 with one constraint and two paths (True and False paths). Each path is modeled with one inequality (if statement) and one equality, the polynomial functionality of the path.

3. PREVIOUS WORK

The goal of this section is to show some theories from computer algebra and software testing that will be used to define the new coverage metric. These theories will be presented and adapted to their application.

3.1 Notation

As has been described in the previous section, the system under verification is described with a set of polynomials. Let *'r'* be the number of variables or dimension of one of these polynomials. Let $X=(x_1,...,x_r)$ be any point in the input space. Two types of input spaces or domains are considered: real (R_r) and *'q'*-element discrete (F_q^r) input domains. Let $R[x_1,...,x_r]$ be the space of all *r*-variable polynomials with real coefficients and $R[x_1,...,x_r]_d$ be the subspace of polynomials of total degree of at most 'd'. This subspace is formed by polynomials

$$P(X) = \sum_i a_i x^i \ . \tag{1}$$

where the coefficients, a_i, are real numbers and $i = (i_1, ..., i_r)$ with $i_1 + ... + i_r \le d$, where i_i is an integer that fulfils $0 \le i_k \le d_k$, $k = 1, ..., r$. Additionally, $x^i = x^{i_1}...x^{i_r}$. For example, the polynomial "$3.1x^2y + 4.2x - 0.7$" is a member of the set $R[x, y]_3$: 2-variable polynomials of total degree of at most 3.

Let $F_q[x_1,...,x_r]_d$ be the sub-space of r-variable polynomials with F_q coefficients and total degree of at most 'd'. The number of polynomials of this sub-space is denoted by $\# F_q[x_1,\ldots,x_r]_d$. Equation (1) also models these polynomials but the coefficients (a_i) and the input variables (x_i) are elements of F_q, in this case. For example, the polynomial "$3xy + 2y^2 + 1$", is a member of the 4-element 2-variable 2-degree polynomial set $F_4[x, y]_2$ and the input variables, (x, y), can only take 4 values: 0,1,2,3.

3.2 Estimation of the number of test benches that completely define the functionality of a path (data flow)

The main goal of this subsection is to present computer algebra theory (Lagrange interpolation) that can be used to estimate the number of test benches that completely verify a polynomial function of a path and the properties that these patterns have to fulfill.

Interpolation is the problem of constructing a function 'p' from a given set of data [17]. The interpolation data are obtained by sampling another function (f), thus both functions coincide in that data set. In this paper, we are interested in interpolation by polynomials, thus the function 'p' will be a multivariate polynomial. This topic is classic in computer algebra and numerical analysis [17][18][19], but we are only interested in some results related to the "Lagrange Interpolation Problem". This problem can be stated in the real domain in the following form:

Given a finite number of R_r points ($X_1,..., X_L$) and some real constraints $y_1,...,y_L$, find a polynomial $p \in R[x_1,...,x_r]_d$, such that:

$$p(X_j) = y_j \qquad j = 1,...,L \tag{2}$$

In particular, we are interested in values of L for which there exists only one polynomial *'p'*. From the verification point of view, this set of L inputs is the maximum number of test benches that has to be applied to a polynomial model in order to guarantee that there is only one possible behavior. Thus, if a polynomial model is affected by a bug or a "functional fault" that modifies the polynomial coefficients but not the polynomial degree (so, the faulty or buggy polynomial is an element of $R[x_1,...,x_r]_d$), there will be points of the previously commented set in which the faulty/buggy polynomial and the correct polynomial take different values. As far as we know, there is no published method that interpolates a multivariable polynomial from a totally random set of points although this is possible in special cases. For example, for real numbers, the interpolation by tensor product (or rectangular grid) is a classic method to obtain a set of L interpolation points [11]. The first step is to select ($d_i + 1$) random values of the variable x_i, $i = 1,..,r$, where d_i is the polynomial degree of variable x_i. Afterwards the tensor product of these values provides the N interpolation points, with

$$L = \prod_{i=1}^{r} (d_i + 1) \ . \tag{3}$$

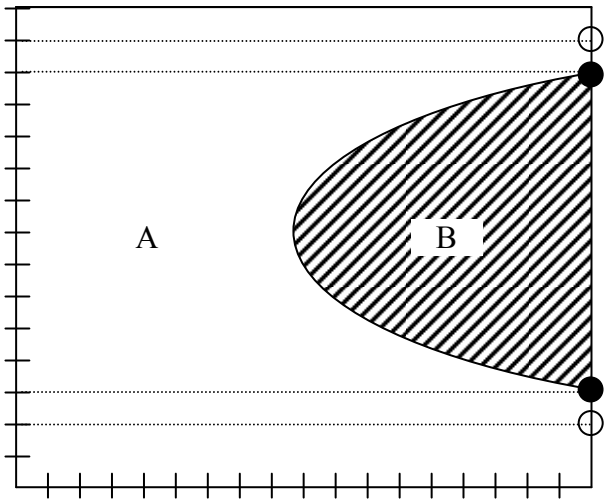
For example, given a system output that is modeled by the polynomial $f(x, y, z) = 8x^2 + 9y + 10z^3 + 9$, it is possible to define a set of "$L = (2+1)\ (1+1)\ (3+1) = 24$" test benches that distinguish this polynomial from any other polynomial, with a degree in (x, y, z) of $(2, 1, 3)$. In order to obtain this set of test benches, it is only necessary to define 3 sets of random values. The first set has three random x values $(x1, x2, x3)$, the second set has two random y values $(y1, y2)$ and the last set has four random z values $(z1, z2, z3, z4)$. The "tensor product" of these sets defines the 24 test benches: $(x1, y1, z1), (x1, y1, z2), ..., (x3, y2, z4)$.

3.3 Estimation of number of test benches that completely define an equation / inequality restriction (control flow)

This section presents the domain testing method inherited from the software testing which is used to check the constraints or boundary conditions. This method establishes a number of test benches by condition. Next, they are explained the limitations and how they are corrected to. Finally, it is defined the number of test benches that completely verifies a polynomial constraints of a path and the properties that these patterns have to fulfil.

The domain testing method, proposed by White and Cohen [10], is a strategy to discover errors in the boundary conditions of software. This strategy consists in generating a set of points on the border of both domains in order to check the limit and detect shifts or changes of the boundary. The work [10] introduces a method of generating this set of points known as N x 1. This strategy requires two types of points: N ON-points lie on the border of closed domain and one OFF-point on the border of open domain. Clarke et al. [14] propose other strategies to generate the points, known as N x N and V x V. The former is similar to the previous strategy, but with N points in both domain and each ON-point has to be as close as possible of an OFF-point. The latter strategy consists of a set of points that corresponds to the vertices of the domains. A simplified strategy is proposed in paper [15], which requires a constant number of points independently of the dimension or the type of the boundary condition or the number of vertices on the given border.

All of these techniques consider linear boundary conditions and/or the possibility of obtaining a very small distance between the ON and OFF points. However, nonlinear boundary conditions are probable in hardware. Moreover, in the integer domain, the minimum distance is limited to one. These properties increase the number of possible errors that cannot be discovered by a set of test benches. Fig. 2 shows an example of the N x N or V x V strategy with nonlinear constraints and integer domains. The stuffed points represent ON points and the hollow points, the OFF points. The set B is the closed domain and set A is the open domain. The problems of this strategy are visible: there are many possible functions that fulfil these conditions.



if ($25x + 9y^2 - 144y + 801 \geq 0$)
  Domain B
else
  Domain A

Fig. 2. Limitations of N x N and V x V strategies.

In order to reduce the possible errors, Lagrange interpolation can also be applied to the restrictions. The proposed strategy is to generate *L x L* points according to formula 3. For example, the case in figure 2 needs 6 ON-points and 6 OFF-points to guarantee that all possible errors are detected.

The main disadvantage of this method is the number of points that must to be generated compared to the previous methods. However, the number of points can be reduced when the domain is defined by several restrictions or when a restriction is shared by several domains. In the former case, the intersection point between two restrictions is the best point because it is common to two restrictions and it is equivalent to two points, one for each restriction. Figure 3 shows this case. The boundary of the domain B is generated by two linear restrictions. It is necessary to generate 2 x 2 points per restriction or 4 points for each domain. However, only three points are generated for each domain. The common points are equivalent to 4 points.

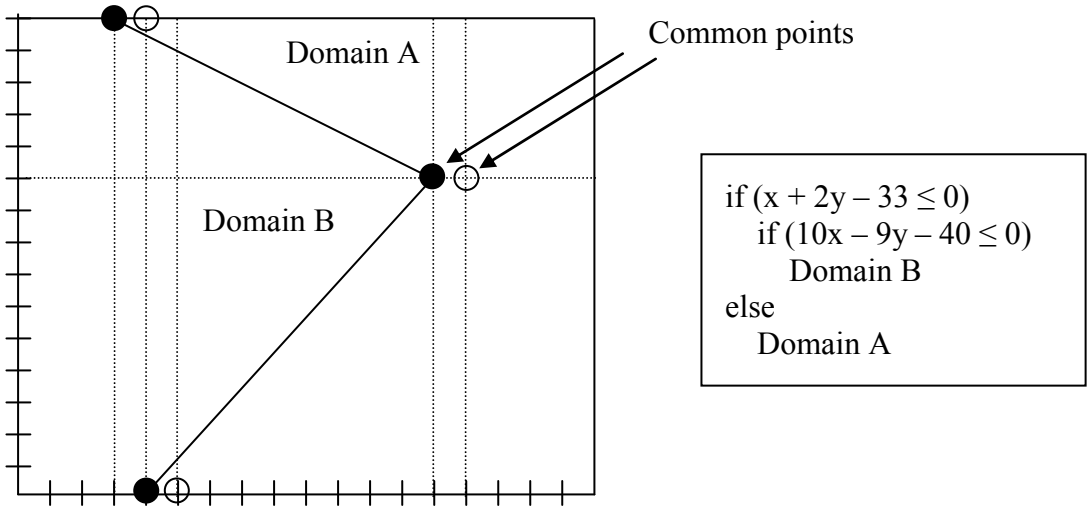


Fig. 3. Optimizations of the test bench generation

In the latter case, the restriction is common to several domains. Therefore, if a domain generates all the points necessary to check the restriction, the other domains do not need to generate more points.

4. COVERAGE METRIC

4.1 Metric

The new coverage metric represents the probability of detecting functional errors in the set of test benches. This metric depends on the number of the path. In this work, the number of path is restricted to a limit of depth established by an input argument of the tool. For each path, it takes into account two parameters: the functionality of the path and the boundary condition of the path. Section 3 shows the number of points needed to reach 100% coverage in functionality and the restrictions of the path. The sum of both numbers defines 100% of path coverage.

The (4) is the equation of the coverage. It is equal to the sum of the coverage of each path divided by the number of paths.

$$Cov = \sum_{each\ path} \frac{\sum_{i} \text{achieved\_point}_i \times w_i}{\sum_{i}^{\text{Total points}} \text{point}_i \times w_i} \times \frac{100}{number\ of\ paths}\% \tag{4}$$

The implemented tool covers each path, generates several points to check the data and control flow and updates the coverage. The false paths are detected and not included in the equation. The points of a path are weighted depending on the number of points obtained. A unchecked restriction is penalized more than a restriction with some points.

4.2 Test benches Selecting Strategy

The objective is to generate enough points for each path so that the polynomial function and the path limits are defined. It is important to observe that the points that check restrictions of a path are valid to check the path function. However, this is not true otherwise. This is the reason why the first step is to generate the points that check the boundary path, and then, the points that check the function of the path. In general, checking the functionality of the path requires fewer points than checking the boundary path. This property enables the reduction of the number of points.

In order to generate the points that check the restrictions, the tool looks for ON-points or OFF-points that fulfil both restrictions. The restrictions of a path are taken in sets of two to search common points: the intersection of both restrictions.

5. EXPERIMENTAL RESULTS

In order to demonstrate the coverage capacity of this new metric, a set of test benches were generated using a new tool and then, the test benches were compared with traditional coverage metrics and mutation testing. Three examples are considered. The first, ALU, is a control and data example. It models the main functionality of PIC17. The second, Triangle, is a control dominated example whose function is the detection of the type of triangle (scalene, isosceles, equilateral and non triangle). Finally, Ellipf is a data dominated example (from HLSW 92 benchmarks). It's a fifth order elliptical wave filter. Some characteristics are shown in the table 1.

	Lines	Branches	Paths	Mutants
ALU	70	20	14	1231
Triangle	50	34	18	951
Ellipf	55	0	0	1103

Table 1: Characteristic of the benchmarks

The tool developed automatically generates a set of test benches. The input is a C function which models reactive processes. This tool searches for the set of test benches with maximum coverage but with the minimum number of ones. The tool does not support pointers. The results are shown in table 2. The second row is the number of test benches generated by the new tool. The next row shows the code coverage (line, branch and path) of the set of test benches. The fourth row corresponds to the new coverage and finally, the last row to the coverage of mutation. This coverage metric is obtained using the Mothra tool [20] which considers 22 mutant types of operators for Fortran 77.

		ALU	**Triangle**	**Ellipf**
Number of test benches		27	22	7
Code coverage	Line coverage	100%	100%	100%
	Branch coverage	100%	100%	100%
	Path coverage	100%	100%	100%
Proposed Coverage		59%	86%	100%
Mutation Coverage		80%	81%	94%

Table 2: Comparison of coverage metrics

These results show the limitation of the code coverage. All the sets of test benches obtain 100% in the coverage metrics (line, branch and path). However, the proposed coverage is less than 100%, except in the ellipf example (data dominated example). The test benches generated obtain good mutation coverage. The mutation coverage is less than the proposed coverage in two cases for two reasons. First, the difficulty of modelling the correct range of the variables in Mothra and second, there are mutant operators that change the order of polynomials, for example, the mutant arithmetic operator replacement (aor) and scalar for constant replacement (scr). The consequence is an increase in the number of test benches, which it is not contemplated in this new coverage.

	Proposed coverage		Mutation coverage	
	Number of test benches	Coverage	Number of test benches	Coverage
ALU	27	80	754	89
Triangle	22	81	664	83
Ellipf	7	94	1083	96

Table 3: Comparison of sets of test benches

Finally, the set of test benches is compared to the set generated by Mothra. Table 3 shows the number and coverage for each set. The number of tests of the new tools is much smaller than for the automatic generator of Mothra but with a similar coverage.

6. CONCLUSIONS

This work introduces a new coverage metric that is closer to the system functionality. It enables the detection of parts of the functionality that are difficult or impossible to detect with classical code coverage. It has coverage percentages that are smaller than the traditional line, branch and path coverage. Although the calculation of the coverage is more complex than code coverage and the generation of the test benches requires more computational effort than random generation, the result is better because the number of test benches is reduced drastically and the coverage metric is closer to the system functionality.

REFERENCES

[1] Beizer, B., "Software testing techniques", Van Nostrand Reinhold, New York, 1983.
[2] Thaker, P.A., Agrawal, V.D. and Zaghloul, M.E., "Register-transfer level fault modeling and test evaluation techniques for VLSI circuits", Proc. ITC 940-9 (2000).
[3] Thaker, P. A., Agrawal, V. D.; Zaghloul, M. E., "Register-transfer level fault modeling and test evaluation techniques for VLSI circuits", IEEE International Test Conference (TC), p. 940-949, 2000.
[4] Ferrandi, F., Fummi, F.; Pravadelli, G., Sciuto, D., "Identification of Design Errors through Functional Testing", IEEE Transactions on Reliability, v.52, nº 4, p. 400-412, December 2003.
[5] King, K.N., Offutt, A. Jefferson, "Fortran language system for mutation-based software testing", Software - Practice and Experience, v.21, nº 7, p. 685-718, Jul 1991
[6] Offutt, A.J., Lee, A.., Rothermel, G., Untch, R. H., Zapf, C., "Experimental determination of sufficient mutant operators", ACM Transactions on Software Engineering and Methodology, v.5, nº 2, p. 99-118, Apr 1996.
[7] Baraza, J.-C., Gracia, J., Blanc, S., Gil, D., Gil, P.-J., "Enhancement of fault injection techniques based on the modification of VHDL code", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, v.16, nº 6, p. 693-706, June 2008.
[8] Fallah, F., Devadas, S., Keutzer, K., "OCCOM - Efficient computation of observability-based code coverage metrics for functional verification", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v.20, nº 8, p 1003-1015, August 2001.
[9] Verma, S., Ramineni, K., Harris, I.G., "A control-oriented coverage metric and its evaluation for hardware designs", Journal of Computer Sciences, v.5, nº 4, p. 302-10, 2009.
[10] White, L.J., Cohen, E.I., "A domain strategy for computer program testing", IEEE Transactions on Software Engineering, v SE-6, n 3, p. 247-57, May 1980.
[11] Qiushuang Z., Harris, I.G., "A domain coverage metric for the validation of behavioral VHDL descriptions", Proc. International Test Conference, p. 302-8, 2000.
[12] Luo C., Yang J., Shi L., Wu X., Zhang Y., "Domain strategy and coverage metric for validation", Proc. 6th International Symposium on Quality Electronic Design, p. 40-5, 2005.
[13] White, L.J., Cohen, E.I., "A domain strategy for computer program testing", IEEE Transactions on Software Engineering, v SE-6, nº 3, p. 247-57, May 1980.
[14] Clarke, L.A., Hassell, J., Richardson, D.J., "A close look at domain testing", IEEE Transactions on Software Engineering, v SE-8, nº 4, p. 380-90, July 1982.
[15] Bingchiang J., Elaine J. W., "A simplified domain-testing strategy", ACM Transactions on Software Engineering and Methodology 1994.
[16] Drechsler R., “Advanced formal verification”, Kluwer Academic Press, 2004.
[17] Gasca, M., Sauer, T., "Polynomial interpolation in several variables", Advances in Computational Mathematics, v.12, nº 4, p 377-410, 2000.
[18] Ar, S., Lipton, R.J., Rubinfeld, R. Sudan, M., "Reconstructing algebraic functions from mixed data," Foundations of Computer Science, 1992. Proc. 33rd Annual Symposium on, vol., no., pp.503-512, 24-27 Oct 1992.

[19] Krick T., "Straight-line Programs in Polynomial Equation Solving", Foundations of Computational Mathematics Conference, 2002.
[20] DeMillo, R.A., Guindi, D.S., McCracken, W.M., Offutt, A.J., King, K.N., "An extended overview of the Mothra software testing environment", Software Testing, Verification, and Analysis, 1988., Proc. of the Second Workshop on , vol., no., pp.142-151, 19-21 Jul 1988